CHAPTER 18

Windows 2000 Security: Advanced Topics

In This Chapter

Learning the practical side of working with Windows 2000 Security features

Defining user rights through group and individual account settings

Creating logon alternatives

Understanding the uses of alternate identification methods such as biometrics

Posing as someone else to perform a specific application task behind the scenes

Security is an important issue today and will continue to grow in importance as everyone gets connected to the Internet. One look at any trade press magazine tells you about a variety of activities by unknown crackers designed to drive network administrators crazy. In fact, at the time of this writing, several companies have been embarrassed by cracker break-ins that have done little more than add a graffiti effect to some of the company's Web pages. It's even more mind boggling to think that some of these efforts have also affected the government and normally "untouchable" organizations like the FBI. Needless to say, the new security features that Windows 2000 provides are an essential reason for companies to upgrade.

For the purposes of this chapter, the term *cracker* always refers to an individual who's breaking into a system on an unauthorized basis. This includes any form of illegal activity on the system. On the other hand, a *hacker* refers to someone who performs low-level system activities, including testing system security. There are times when you need to employ the services of a good hacker to test the security measures you have in place, or suffer the consequences of a break-in. This chapter uses the term *hacker* to refer to someone who performs these legal forms of service.

Chapter 17 takes you on a brief tour of the underlying security features that Windows 2000 provides from an architectural perspective. This chapter takes a somewhat different direction. You also look at some of the new programmer-specific features that Windows 2000 provides to create applications. Although this chapter doesn't even scratch the surface of the new security features that Windows 2000 provides, it does give you enough information to start writing your first Windows 2000 applications that include the new security features.

The first section of this chapter spends some time looking at the Windows 2000 security API in general. We'll look at the various commands you can use to perform general security tasks. Obviously, general security includes working with users and groups, as well as objects. Normally, the system determines what resources and objects a user can access automatically, so you don't have to worry about what that level of access is as long as it's sufficient to protect your application. However, there are situations when you do need to know what access the user has, or even set up new levels of access. For example, when an application is first installed, Windows 2000 provides a default level of security based on the user's current access, which may not be sufficient to perform the task at hand.

The first section also looks at one of the few components that you really need to know about when it comes to security concerns. Microsoft does provide you with access to a few security components within Windows 2000 (and you can be sure there will be more in future versions of the product). The most important of these components allows you to display the Access Control Editor, the same set of property sheets used in several places in Windows 2000 to allow you to set the security associated with a particular item. In fact, there are two sets of these property sheets. The first provides simple access to security, whereas the second provides advanced administrator-level security. We look at both sets of property sheets in this section of the chapter.

In the next section of the chapter, we'll look at the first line of defense when it comes to ensuring your applications are safe — the Windows logon. Microsoft realizes that your company may have special logon requirements that aren't handled by the standard logon dialog box. We'll talk about ways that you can implement logon alternatives such as retinal scans or fingerprint identification. Many companies use physical feature logon (biometrics) rather than a name and password because physical features like fingerprints and voice prints are much harder to steal and therefore safer than a password.

Working with Windows 2000 Security

Security is one of the few places where Microsoft hasn't made much of an impact with COM components yet. In fact, this is one of the places where you definitely need a good working knowledge of the API method of interacting with Windows. Based on Microsoft's current trend in reworking Windows, you can expect this requirement to change sometime in the future. But for now, C++ programmers still have a definite advantage in working with Windows when it comes to security.

Obviously, there's nothing preventing you from creating your own security components that encapsulate your company policies in an easy-to-use form. In fact, this is one of the ways that you can ensure that programming teams within your company implement security in precisely the same way. Using components to access the underlying API gives you the flexibility required to make tough policy decisions one time — and still ease the programming burden of fellow programmers.

In Chapter 17, I talk about all of the architectural details that make up Windows 2000 security. Before you read this section, make sure you know how Access Control Lists (ACLs) and Access Control Entries (ACEs) work. It's also important to know the difference between the discretionary ACL (DACL) and the system ACL (SACL). You should know how group and individual security attributes get attached to various objects and what tokens are used for when it comes to object access. If any of these terms are unfamiliar, you may want to spend some time in Chapter 17 before proceeding with this chapter.

The computer industry as a whole is constantly coming up with new Application Programming Interfaces (APIs) for Windows 2000. One of the latest security APIs has to do with biometrics, the use of human body elements such as the iris and fingerprints for identification purposes. (See the "Identification Alternatives to Logging In" sidebar in this chapter for more details.) The *Biometrics API (BAPI)* allows programmers to embed biometric technology into their applications. BAPI is being put together by a consortium of vendors, including IBM, Microsoft, Compaq, IO Software, Sony, Toshiba, and Novell. One place to learn more about BAPI is at the IO Software Web Site at `http://www.iosoftware.com/bapi/index.htm`. This site allows you to download an overview, general information, technical information, and the BAPI Software Development Kit (SDK).

In addition to BAPI, there are two other APIs under development. The *Human Authentication API (HA-API)*, which was introduced by National Registry, Inc., and *Speaker Verification API (SVAPI)*, which you can find at `http://www.srapi.com/svapi/`, are two specialty biometric APIs. Both APIs are a higher level than BAPI and tend to deal with a specific kind of biometric authentication, rather than biometric authentication in general. Lest you think that all of these APIs are vendor specific, you can also find a central biometrics-standards Web site at the Biometrics Consortium Web site at `http://www.biometrics.org/`.

I'll talk about two essential topics in this portion of the chapter. The first is programmer specific. Though users may be faintly aware that there's a security API, they're unlikely to use it or even care that it exists. As a programmer, you need to be very aware of this part of Windows 2000 and know how to use the various API calls to make your applications secure.

The second section is user oriented. In fact, this particular feature is already implemented in many areas of Windows. Microsoft has made two security property pages available that you can use to allow an administrator to set security for the objects that you create for your applications. Think of these security property pages as you would the common controls provided by Windows already. They allow you to give your application's security features a consistent look that administrators really appreciate.

Improving Security with Ethical Hacking

Some network administrators are justifiably worried about the security status of their networks, especially with today's Internet connectivity. In some cases, these administrators are literally forcing the users they serve into straight jackets that reduce overall productivity. On the other hand, there are some network administrators who don't worry much about security at all, but really should. In many cases, these network administrators rely on hardware and software solutions to guarantee network security. Unfortunately, network security is all too often just a mirage on the horizon.

The main problem with network security is that it's untested in most cases. Network administrators may not really know the status of their networks or how safe the software and hardware solutions they use really are. Obviously, short of inviting a cracker onto the network to see what he or she can do, there aren't a lot of real-world testing methodologies in place right now.

There are other problems lurking in the background as well. Consider the pharmaceutical company that didn't even think it had any connections to the Internet because such connections were deemed too unsafe for the confidential nature of the company's work. A cracker still managed to get through network security to steal information that the company tried to hide. What was the source of the network intrusion? A user brought a modem from home and installed it on his work machine so he could work on the Internet during the day. Obviously, no amount of security planning in the world could help a network administrator get around this problem.

Crackers are always finding new ways to get around even the best security measures. For example, one company had a very robust security system that included filters, firewalls, and constant security checks. The network administrator was justifiably proud of the amount of work that went into the security system. Unfortunately, crackers still got into the system. The problem was a Unix server that hadn't gotten a bug update. The crackers used the bug in the system to circumvent all of the security measures the network administrator put in place.

There are several things you can learn from these tales of woe. The following list provides you with some things you need to look for when implementing a real-world security system:

- ✦ Any security system can be circumvented.
- ✦ The humans who use the network are also the greatest threat to its security.
- ✦ Every security system requires constant monitoring and updates.
- ✦ Nonsecurity components of a system are just as important as the security components—make sure you install all required bug fixes.
- ✦ Every network should be tested completely.

This last bullet is what ethical hacking is all about. You can't really test network security unless you invite someone to try and break it. Since the crackers of the world can't be trusted, you need to find someone with the same skills who is willing to use them for ethical purposes. In other words, you need an ethical hacker (or a vulnerability tester, as some companies prefer to call them). Ethical hackers help network administrators create a more secure environment by testing networks for potential security flaws. They help you find the unknown modem or the nonupdated server that can cause problems in the future.

Ethical hackers can do more than just help you find the weaknesses in your defenses; they can help convince management of the need for additional security measures as well. Sometimes the problem with security isn't a lack of concern on the part of the network administrator or a lack of awareness on the part of the user; it's a lack of resources required to make the required security updates. In other words, the services of an ethical hacker can prevent management from becoming their own worst enemy.

There's one problem with this whole scenario: Just where do you find an ethical hacker? ICSA is a security consulting company whose main concern is helping you get the most secure setup for your network. You can contact them at `http://www.icsa.net/`. This site contains a wealth of resources for administrators who want to make their networks just a little more secure. There's also an Intrusion Detection Systems Consortium that was recently formed to discuss the various security issues facing network managers today. Current members of the consortium include AXENT Technologies, Inc., Centrax Corporation, Internet Security Systems, Inc., Internet Tools, Inc., Memco Software, Inc., Netect, Inc., Network Associates, Inc., Security Dynamics, Inc., and Tripwire Security Systems, Inc. You can find out about this group at `http://www.icsa.net/news/press_room/1998/idsc.shtml`.

Windows 2000 Security API Overview

Before you can do much at all with Windows 2000 security, you need to know something about the security API. Let's begin with the user end of the picture. It's important to understand that the user's access is limited to the combination of groups and individual rights that the administrator assigns. However, most of the configuration options available to the administrator affect Windows as a whole, not just your particular application. If you want the administrator to set user-level access for your application in particular, then you need to provide the ability to set individual user access for each object or task your application provides.

User level access depends on a security ID (SID). When the user first logs into the system, Windows assigns an access token to the user and places the user's SID (stored on the domain controller or other security database) within it. The user object carries both the access token and the SID around for the duration of the session. An access token also contains both a DACL and an SACL. The combination of ACLs and SID within the access token is a key that allows the user access to certain system resources. Just like any key, the user's access token is unlikely to fit every lock.

As a result, you have to change the access token to fit within the locks of your application, but only those locks that the administrator actually wants the user to open. Table 18-1 contains a list of the various API functions that you'll commonly use to change the user's access token. This list only provides an overview, and not a detailed description, of each API function.

The tables in this section don't provide a complete list of all of the functions in the security API. I chose these functions because they're unique, new, modified, or a representative example of a larger group of function within Windows 2000. These are also the functions that you may need to use most often to secure your applications.

Table 18-1
Common User Access Token Function Overview

Function Name	*Description*
AdjustTokenGroups	Allows you to adjust one or more group flags that control how one or more groups are used within the access token. For example, you may want to make a particular user the owner of a group in place of the current owner.
AdjustTokenPrivileges	Allows you to adjust one or more privileges within the access token. All you can do with this function is enable or disable an existing privilege; you can't add or delete privileges from the access token.
AllocateLocallyUniqueId	Creates a new LUID. The LUID is only unique for the current computer session and can be duplicated on other computers or after the current session is ended (normally after the computer is rebooted). Unlike a GUID, a LUID is temporary.
BuildExplicitAccessWithName	Creates an EXPLICIT_ACCESS data structure for the named trustee. An EXPLICIT_ACCESS data structure defines the ACL information for the trustee. You can use this data structure with API functions such as SetEntriesInAcl to define a trustee's access level to objects. The EXPLICIT_ACCESS data structure can affect either the SACL or DACL depending on the access mode you set for it.
BuildTrusteeWithName	Creates a TRUSTEE data structure that's used to identify a specific trustee. You supply a trustee name and Windows fills the other data structure elements with default values. This means you need to modify the data structure before using it.

Function Name	*Description*
BuildTrusteeWithSid	Creates a TRUSTEE data structure that relies on a SID rather than a trustee name. The default data structure values are modified appropriately.
CheckTokenMembership	Checks whether a specific SID is enabled within an access token. This, in turn, can help you to determine whether a user or process belongs to a particular group.
CreateRestrictedToken	Creates a duplicate of an existing token. The new token has only a subset of the rights within the existing token. You can't use this function to add new rights to the resulting token.
DuplicateToken	Creates a copy of an existing token. Using this technique allows you to create a new token that varies from an existing token by one or two privileges.
DuplicateTokenEx	Creates a duplicate of a token. However, this function allows you to create either a primary or impersonation token. In addition, you can set access rights to the new token as part of the duplication call.
GetAuditedPermissionsFromAcl	Returns a list of ACL entries that result in an audit log entry for the specified trustee. This includes ACL entries that directly affect the trustee as well as groups to which the trustee belongs. In other words, you get a complete list of all audit-generating access events—not just those associated with the specific trustee. The audited access list is returned in an ACCESS_MASK data structure.
GetEffectiveRightsFromAcl	Returns a list of ACL entries that list the effective rights for the specified trustee. These effective rights are returned in an ACCESS_MASK data structure.
GetExplicitEntriesFromAcl	Returns an array of EXPLICIT_ACCESS data structures that define the level of access the trustee is granted by each ACE within an ACL. The data structure provides information such as the access mode, access rights, and inheritance setting for each ACE.
GetTokenInformation	Returns a data structure containing complete information about the access token. This includes information such as the token's user, groups that appear within the token, the owner of the token, the impersonation level, and statistics associated with the token.
GetTrusteeForm	Returns a constant value that equates to one of the TRUSTEE_FORM enumeration values for a given trustee. In most cases, these constants indicate whether the trustee is a name, SID, or object.

Continued

Table 18-1 *(Continued)*

Function Name	***Description***
GetTrusteeName	Returns the name associate with a name trustee. If the TRUSTEE data structure that you provide is for a SID or object, the return value is NULL.
GetTrusteeType	Returns a constant value that equates to one of the TRUSTEE_TYPE enumeration values for a given trustee. In most cases, these constants indicate whether a trustee is a user, group, domain, or alias. There are also values to indicate whether the trustee is deleted or invalid.
IsTokenRestricted	Detects whether the access token contains one or more restricting SIDs.
LookupPrivilegeDisplayName	Allows you to convert a privilege name as listed in WINNT.H to human-readable form. For example, the SE_REMOTE_SHUTDOWN_NAME constant can convert to the human-readable form of "Force shutdown from a remote system."
LookupPrivilegeName	Allows you to convert a privilege name specified by a LUID to one of the constant forms listed in WINNT.H.
LookupPrivilegeValue	Allows you to convert a privilege name as listed in WINNT.H to a LUID.
OpenProcessToken	Open a token that's associated with a process (application). Like a file, you need to specify what level of access you want to the token. For example, there's a constant that tells Windows that you only want to work with the token's groups. The TOKEN_ALL_ACCESS constant gives you complete access to the token.
OpenThreadToken	Opens a token that's associated with a thread within an application. As with a process token, you need to request a specific level of access when making the request.
SetEntriesInAcl	Creates a new ACL by merging new access control or audit control information into an existing ACL. You can use this function to create an entirely new ACL by creating the ACL using the ACL creation function, BuildExplicitAccessWithName.
SetThreadToken	Used mainly to implement impersonation within a thread. You can use this function to give different rights to a single thread within an application. This, in turn, allows that thread to perform tasks that the user may not ordinarily have the rights to perform. I'll talk about this subject more in the "Overriding Native User Security to Perform Specific Tasks" section of this chapter.

Function Name	*Description*
SetTokenInformation	Allows you to set the information contained within an access token. However, before you can set the information within the token, you have to have the required access rights. There are three data structures associated with this function that allow you to adjust the owner, primary group, and DACL information.

Not every security API function is supported by every version of Windows currently on the market. Windows 95/98 provides far less support than Windows NT/2000 does for security. Windows CE provides even less support than Windows 95/98 does. In short, write your application so that it takes the ability of the operating system to react to the security API request into account. Fortunately, there are few cases when requesting a level of support that a version of Windows can't provide results in application failure. For example, Windows 95/98 normally returns a success value if it doesn't support a particular security API function so that the application can continue processing the request. Obviously, this means that you have to check the operating system that the application executes on an act accordingly.

Normally, you never work with SIDs directly. The reason is that you can address a user by their login name and make your code easier to debug and understand. However, there are certain situations when you may want to work with SIDs. The most important of these situations is when you're dealing with common SIDs such as the one for the World, which has a SID of S-1-1-0. The SID for the World always remains the same, but the name for the World can change from country to country. As a result, you should always refer to common, universal SIDs by their SID rather than a common name. With this in mind, you should know a bit about the SID-related functions for those times when you want to work with common SIDs. Table 18-2 contains a list of SID-related functions.

Table 18-2
Common SID-related Functions Overview

Function Name	*Description*
AllocateAndInitializeSid	Allows you to create and initialize a SID with up to eight subauthorities.
ConvertSidToStringSid	Converts a SID to a string in human-readable format. This format consists of a minimum of four values in the form "S-R-I-SA", where S designates the string as a SID, R is the revision level, I is the identifier authority value, and SA is one or more subauthority values. Note the dashes between SID values—these dashes are always part of any human-readable SID string.
ConvertStringSidToSid	Converts a specially formatted string into a SID.

Continued

Table 18-2 *(Continued)*

Function Name	***Description***
CopySid	Creates a duplicate of an existing SID.
EqualPrefixSid	Compares two SID prefixes for equality. A SID prefix is the SID value minus the last subauthority value. This test is useful for detecting two SIDs that are in the same domain.
EqualSid	Compares two SIDs for equality in their entirety.
FreeSid	Deallocates the memory used by a SID previously created using the AllocateAndInitializeSid function.
GetLengthSid	Returns the length of a SID in bytes.
GetSidIdentifierAuthority	Returns a pointer to a SID_IDENTIFIER_AUTHORITY data structure. This data structure contains an array of six bytes that specifies the SID's top-level authority. There are several predefined authorities, including NULL (0), local (1), world (2), creator (3), and Windows NT/Windows 2000 (5).
GetSidLengthRequired	Returns the length of a buffer required to hold a SID structure with a specified number of subauthorities.
GetSidSubAuthority	Returns the address of a specific subauthority within a SID structure. The subauthority is a relative identifier (RID).
GetSidSubAuthorityCount	Returns the address of a field within a SID structure used to hold the number of subauthorities for the SID. You can use this address to determine the number of subauthorities within the SID.
InitializeSid	Sets the identifier authority of a SID structure to a known value using a SID_IDENTIFIER_AUTHORITY data structure. The values of any subauthorities aren't set using this function. Normally, you want to use the AllocateAndInitializeSid function to perform a complete SID initialization.
IsValidSid	Determines if the contents of a SID structure are valid. This includes checking the revision number and ensuring that the number of subauthorities doesn't exceed the maximum value.
LookupAccountName	Retrieves the SID (and accompanying data) for a specific account. You must supply an account and system name where the account can be found.
LookupAccountSid	Retrieves the name and machine associated with a given SID. This function also returns the name of the first domain where the SID is found.

Obviously, security isn't this one-sided. Once Windows has determined what rights the user or other object has, it must match those rights to the access requirements of the system resource that the user or object is requesting to access. This means working with security descriptors. A *security descriptor* essentially is a lock on the door of the object or other system resource. The key (access token) either fits the lock or it doesn't. Access is granted or denied based on how the key fits the lock. Table 18-3 presents an overview of the API functions at your disposal for working with security descriptors.

Table 18-3
Security Descriptor Function Overview

Function Name	*Description*
ConvertSecurityDescriptorToStringSecurityDescriptor	Converts a security descriptor into a string format. The flags you provide as part of the call determine the amount of information you get in the string. A complete string contains the owner SID in string format, the group SID in string format, a DACL flag list using coded letters, a SACL flag list using coded letters, and a series of ACE entries in string format.
ConvertStringSecurityDescriptorToSecurityDescriptor	Converts a specially formatted string into a security descriptor.
GetNamedSecurityInfo	Returns the security descriptor for the named object you provide as input. A series of flags determines what kind of information to retrieve.
GetSecurityDescriptorControl	Returns the security descriptor control information and revision number for the security descriptor structure you provide as input.
GetSecurityInfo	Returns the security descriptor for an object that you specify using an object handle. Windows 2000 provides flags that determine which security descriptor entries get retrieved.

Continued

Table 18-3 *(continued)*

Function Name	*Description*
SetNamedSecurityInfo	Modifies the security descriptor information for an object that you specify by name.
SetSecurityDescriptorControl	Modifies the control bits of a security descriptor. There are other functions related to this one, such as SetSecurityDescriptorDacl, that allow you to set other control bits of the security descriptor. This particular function is new to Windows 2000.
SetSecurityInfo	Modifies the security information within the security descriptor for an object. You can use this function to modify the owner, group, SACL, or DACL. Each information type requires a separate data structure; you use flags to tell Windows 2000 which elements to change. The object is identified through a handle and object type descriptor.

By now, you should have some idea of how to work within the security architecture that Windows 2000 provides. The divisions that I set up within the various tables are artificial; they're for description purposes to make the functions easier to comprehend and use. In a real-world application, you're likely to combine elements of all three tables to create a complete security picture.

Using the Access Control Editor

The Access Control Editor is a COM control that allows you to add a standard interface to your application for the sole purpose of allowing administrators to set application security as needed. These are the same property pages that Microsoft uses within Windows 2000 to set various kinds of security. There are two sets of property pages provided by the Access Control Editor. Figure 18-1 shows a sample of the dialog box that the user sees for the simple property page. (I chose this particular example so that you can see the dialog box in action.) Figure 18-2 shows the advanced property page. As you can see, both property pages allow the administrator to work with the security settings for an application with relative ease.

Figure 18-1: The Security tab on a drive Properties dialog box starts with the simple version of the Access Control Editor.

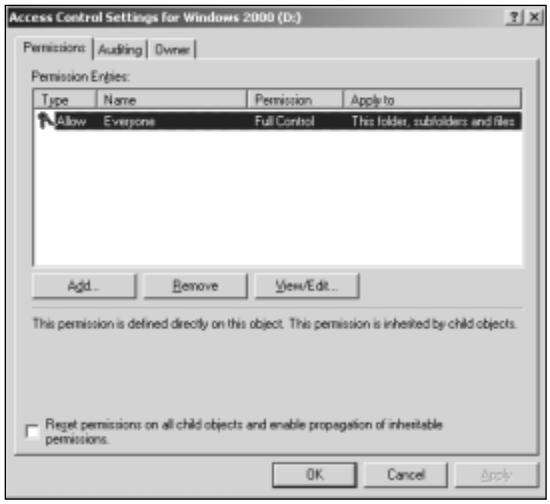

Figure 18-2: Clicking the Advanced button displays this more complex version of the Access Control Editor.

Notice that the advanced dialog box provides very complete controls for setting every aspect of security for this particular object (a disk drive on my workstation). The first tab (Permissions) is designed to set the DACL, the second (Auditing) the SACL, and the third (Owner) the owner information. The only thing that's missing is the group information, which really isn't important at the user level in many cases.

Now that we've gotten past the user interface element of the Access Control Editor, let's look at the COM component part. You create the Access Control Editor property sheets using the ISecurityInformation interface. There are two main methods used to call on this interface: CreateSecurityPage( ) and EditSecurity( ). The CreateSecurityPage( ) method is used to add a Security tab to an existing property sheet. You can use the PropertySheet( ) function or the PSM_ADDPAGE message to add the resulting property page to the existing property sheet. The EditSecurity( ) method displays the Security property page on a property sheet that's created specifically to display the Access Control Editor.

There's some two-way communication that takes place between your application and the result Access Control Editor. For one thing, the Access Control Editor doesn't know anything at all about your application when you first start it, so you have to supply this information. In short, you need to implement several methods of the ISecurityInformation interface within your application. The following list provides a very brief description of these functions:

GetObjectInformation( )	Obtains the information required to initialize the Access Control Editor
GetSecurity( )	Obtains the current security descriptor of the object being modified
SetSecurity( )	Returns an updated security descriptor to the object after the user completes the editing process or when Apply is pressed on the Access Control Editor dialog box
GetAccessRights( )	Obtains the access rights of the object being edited
MapGeneric( )	Asks the object to map the generic rights in an access mask to the standard and specific rights equivalents
GetInheritTypes( )	Requests information on how child objects can inherit the ACEs owned by the requesting object
PropertySheetPageCallback( )	Tells the object that the property page is about to be created or destroyed

The Access Control Editor provides you with a lot of control over the security environment it creates. In fact, you can set one of the flags to allow the user to only view the security information for an object. This means that a user can look at what rights they have to an object in your application, then request help from an administrator if needed to make sure that they can perform the required application task.

The implementation of the GetObjectInformation( ) method is especially important because you create an SI_OBJECT_INFO data structure and pass it to the Access Control Editor. This data structure includes a lot of configuration information about the appearance of the security dialog box. For example, you can choose which features the user gets to use. If you only want the administrator to have simple access to the object's security features, you can choose to disable the Advanced button shown in Figure 18-1, making it impossible to change the auditing options or the object owner. In addition, this data structure defines property page elements such as the title bar contents, the name of the object, and the name of a server that the Access Control Editor can use to look up security information about the object.

Working with Windows Logon

Most people take the logon dialog box for granted. It becomes almost second nature to type a name and password when you first start the system. In fact, about the only time that most users actually notice the Windows logon dialog box is when they mistype their password or the administrator revokes their access to the system.

This familiarity with the Widows logon dialog box has created some problems for network administrators. For one thing, passwords aren't exactly secure. Just about every company has one user (and probably many more) who record their logon password, then leave the password where anyone can see it. In short, passwords often fail because the human element in the security equation fails to take appropriate protective measures like using long and difficult-to-remember passwords.

At the same time that users are getting lax when it comes to passwords, crackers are getting more proficient at barreling their way past firewalls and other forms of protection on the network. As a result, break-ins have become more common; common enough, in fact, that even the FBI experienced a recent security problem on their Web server. With this in mind, Microsoft provides four technologies that a developer can use to enhance both workstation and server security on a network. The following list provides an overview of all four technologies. We'll explore these technologies in the sections that follow.

- Password Filters: Allows password policy enforcement and password notification
- Winlogon User Interface: Provides interactive logon support
- Network Provider API: Allows communication between the Multiple Provider Router (MPR) and the network providers
- Local Security Authority (LSA) Authentication: Authenticates and logs the user into the system

Password Filters

One of the major problems that administrators have to deal with is getting users to create passwords that are difficult to guess, yet easy enough to remember that the user doesn't feel forced to write it down. For example, many administrators use a password policy of a word, followed by a number or special character, followed by another word. While this policy can't keep crackers at bay forever, it does slow them down because the methods required to figure the password out are more complex. In addition, two words with a number or special symbol in-between is relatively easy to remember compared to the arcane jumble of digits and numbers that some administrators try to use.

Unfortunately, Windows doesn't provide any way to implement such a policy through configuration or other standard features. However, there's a way to provide this feature by writing a password filter. Essentially, this is a DLL that gets called as part of the logon process or any other time a user needs to enter a new password.

Every time a user wants to change his or her password, the LSA calls each password filter twice. The first time the filter gets called is to verify the new password. This is where your filter gets a chance to see what the user wants to use for logging into the system. If the password doesn't meet the corporate policy, you have an opportunity to request that the user change the password. The second time the filter gets called is after the new password is implemented. The LSA calls each password filter in turn to provide notification of the password change.

Microsoft provides a default password filter, PASSFILT.DLL. This filter ensures that passwords are at least six characters long; contain a combination of uppercase letters, lowercase letters, numbers, and punctuation marks (you only have to use three out of the four categories); and don't include the user's name. If this is all you need in the way of password verification, then the work is already finished. However, you also can create a password filter of your own. In fact, you can use this default filter and a custom filter in combination.

No matter which route you take, you need to install the filter in the SYSTEM32 folder of the client machine. Once you copy the file, add a registry entry to the HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa\Notification Packages key. (This is the Windows 2000 specific location.) The easiest way to find this key is to use the Edit | Find command to look for a registry key with the value of Lsa. If you don't see the Notification Packages key, then add it. All this means is there aren't any other notification packages installed on the client machine. The last thing you do is add the name of the filter as a string value to the key. You can add more than one password filter to the Notification Package key. Each filter gets called in turn to verify the password added by the user.

Creating a password filter is a fairly simple matter. All you need to do is create a standard DLL project. The DLL you create has to export three functions. The following list provides a brief description of each function.

InitializeChangeNotify( )	This function returns TRUE if the password filter DLL is loaded; FALSE if it isn't.
PasswordChangeNotify( )	This function is called by the LSA after a password change is made. The password filter receives, as input, the username, RID, and new password. The default return value is STATUS_SUCCESS.
PasswordFilter( )	This function is called by the LSA prior to a password change so that the password filter can verify the new password. The LSA provides the account name, username, new password, and operation type as input. The operation type is set to TRUE if the password is being set (normal for a new user) rather than changed. A return value of TRUE indicates that the password is good; FALSE indicates t hat the user needs to change the password.

Winlogon User Interface

The standard Windows logon dialog box that we're all used to seeing is the one that appears when Windows first starts. Windows registers the Ctrl-Alt-Delete key sequence as the one that initializes Winlogon, which means that Winlogon is used in more places than the initial logon dialog box. In fact, these places are defined through the desktops that Winlogon creates as part of the Windows initialization process: Winlogon, application, and screen saver. All three desktops are protected by any access except for Winlogon and protected system processes. So, when users press Ctrl-Alt-Delete and see the Windows Security dialog box, what they're actually seeing is the Winlogon desktop.

The DLL associated with Winlogon is GINA.DLL. (MSGINA.DLL is the default GINA.DLL in Windows 2000.) GINA stands for graphical identification and authentication. This is the application that actually performs the logon process; you can replace it with your own version of the DLL. For example, you may want to create a version of GINA that polls a smart card reader or a retinal scanner instead of presenting the usual logon dialog box.

Adding your custom GINA.DLL to Windows is fairly easy. All you need to do is add a value named GinaDLL to the HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon key. This value should contain the name of your GinaDLL (normally GINA.DLL).

Since the Winlogon.EXE and GINA.DLL are called before there's a session you can use for debugging purposes, Microsoft provides a special way for you to enter debug mode when working with this kind of DLL. All you need to do is add a heading to WIN.INI named [WinlogonDebug]. There are a variety of switches that you can use to debug your GINA.DLL, including one that allows you to bypass the normal Winlogon process. (You need this back door in case the DLL fails to function in order to change the registry settings.)

When creating a GINA.DLL, you use a standard DLL project. A GINA.DLL must export specific functions because Winlogon.EXE calls those functions during the logon process (and at other times when the system needs to verify the identity of the user). Table 18-4 provides a brief overview of the various functions that a GINA.DLL must export.

Table 18-4
GINA.DLL Exported Functions

Function Name	*Description*
WlxActivateUserShell	Starts the user shell program, which is normally Explorer.EXE.
WlxDisplayLockedNotice	Displays a message telling the user that the workstation is in the locked state. This message can include information about the person locking the workstation and when he or she locked it.
WlxDisplaySASNotice	Displays a message and dialog box showing that no one is logged into the machine. You can include instructions for logging into the machine. For example, you want to tell the user to swipe his or her security card through the smart card reader if you choose to use that system.
WlxInitialize	Initializes GINA.DLL for use. This function gets called once for each window station present on the computer.
WlxIsLockOk	Verifies that the workstation lock is still functional.
WlxIsLogoffOk	Verifies that it's all right to log the user off.
WlxLoggedOnSAS	Called when a security attention sequence (SAS) occurs when the user is logged into the workstation and the workstation isn't locked. For example, this function gets called if the user presses Ctrl-Alt-Delete or if he or she inserts a smart card into the smart card reader.
WlxLoggedOutSAS	Called when a SAS occurs when there isn't any user logged into the workstation.
WlxLogoff	Indicates that a logoff sequence was requested by the user or initiated by the system.

Function Name	*Description*
WlxNegotiate	Indicates whether the current version of Winlogon.EXE can be used with the GINA.DLL that's loaded.
WlxNetworkProviderLoad	Called when Winlogon requires user authentication information from GINA.DLL. Winlogon passes this information along to the network provider for verification purposes.
WlxRemoveStatusMessage	Returns a status value indicating whether a status message was removed for the window station provided by Winlogon.
WlxScreensaverNotify	Allows GINA.DLL to interact with the screen saver right before the screen saver becomes active so that it can set security as required.
WlxShutdown	Called right before the system shuts down. This function allows GINA.DLL to perform any required shutdown tasks such as ejecting a smart card from a smart card reader.
WlxStartApplication	example, GINA.DLL is used to restart Explorer.EXE if the application stops unexpectedly. This function also gets called if the system needs to run the extended task manager. Theoretically, this function should be able to run any application on the user's machine.
WlxWkstaLockedSAS	Called when a SAS occurs when the workstation is locked (even if a user is logged into the system).

Network Provider API

There are a variety of network provider types; talking about all of them lies outside the scope of this book. However, there's a special credential manager provider that I can talk about here. The main reason to implement such a network provider is to allow an extra check of the user's credentials beyond the checks provided by the network specific providers (like the ones that a user uses to access a Windows 2000 or NetWare network). Full-fledged network providers perform a wide range of tasks, including device redirection, enumeration of resources, connection-related activities, administrative functions, and credential checks.

A network provider is implemented as a DLL. There are two types of network provider DLLs that Windows 2000 currently recognizes: full network provider and credential manager. A credential manager only needs to export two functions as shown in the list that follows.

NPLogonNotify( )	This function is called to notify the credential manager of a logon event. The credential manager supplies the MPR with a logon script.
NPPasswordChangeNotify( )	This function notifies the credential manager of a change in authentication information. Various information — including the username and password — is provided as input. The credential manager replies with any of a number of responses, including WN_SUCCESS, which means that the authentication update was successful.

LSA Authentication

The Logical Security Authority (LSA) and GINA work hand-in-hand. The functions provided by this part of the security API allow GINA to verify user identities and log them into the system. LSA is connected to quite a few areas of the system. For example, all of the Security Support Providers (SSPs) work with LSA and use it as a conduit when working with GINA. In short, LSA is really a lot of glue code that holds many of the security elements in Windows 2000 together. As a result, I'll talk only about the functions that are directly related to GINA in this section. Table 18-5 contains a list of the functions that you can call within a GINA.DLL.

Table 18-5
GINA Support Functions within LSA

Function Name	*Description*
LsaRegisterLogonProcess	Verifies the caller is a legitimate logon application after connecting to the LSA server.
LsaLogonUser	Verifies the identity of a user. If the user authentication information is valid, then LSA creates a new user session and returns an access token. Otherwise, the LSA returns an error code, which should result in some kind of error message dialog box and a request to supply a valid username and password.
LsaLookupAuthenticationPackage	Returns the identifier for an authentication package.
LsaFreeReturnBuffer	Frees the memory used by a buffer.
LsaCallAuthenticationPackage	Allows a logon application to communicate with an authentication package. You normally use this function to run dispatch routines provided by the authentication package.

Function Name	*Description*
LsaDeregisterLogonProcess	Deletes a logon application context. This is normally called after the application has run and no longer requires access to system resources.
LsaConnectUntrusted	Creates an untrusted connection to the LSA server. The untrusted connection has limited rights on the LSA server.

Identification Alternatives to Logging In

A user name and password is normally the only form of access identification that a user needs to gain access to your server. In most cases, that's all you really need to ensure that the data on your system remains secure. However, even if you make the software that grants access to the system totally bulletproof (an ever more difficult task as computers gain processing power), problems still remain with this approach.

Obviously, the most common problem is lost passwords. Asking the user to provide a password that's difficult to break often results in a password that's also difficult to remember. As a result, the network administrator ends up spending more time fixing lost password problems.

Passwords can also be compromised in a number of ways. The least subtle scenario is the user who writes his or her password down on a notepad, then places the note near the keyboard or monitor. Believe it or not, this happens on a regular basis. At one company, I found a single notepad that contained not only the administrative assistant's password, but the manager's password as well.

Even if everyone at a company does their best to protect security, and you have the best security available for your application, crackers often find ways around the security measures you have in place. Just the fact that your security relies on a password means that someone can guess what that password is given enough time and opportunity. In other words, using the name and password method of security is problematic at best. You can't rely on a password to provide total security of some types of very confidential data.

Some companies are combating these problems using a variety of methods. One of the more common methods at the moment is the use of smart cards. A smart card is a credit card–sized device with some built-in processing power. The user swipes the card in a special reader to gain access to the network. What happens is that the card provides a digital certificate that identifies the user in lieu of a password. One of the nice things about using a smart card is that you don't have to provide any identification like the numbers used by credit cards on the outside.

(Continued)

One company that's starting to make smart card readers a standard option is Hewlett-Packard (HP). Their Vectra Desktop PCs and Kayak PC workstations are now optimized to use either the Gemplus (a Veridicom product) or Schlumberger smart card readers. In addition, the ProtectTools corporate security strategy offered by HP extends from the client to the firewall, which allows a company to give outside partners secure access to company data. The HP TopTools Management software allows managers to view clients that have smart card readers. In essence, there's already an infrastructure and products in place to allow a company to implement full security using smart cards.

A smart card does eliminate the problem of compromised passwords—at least without assistance. Without a password lying around, it takes a concerted effort by an inside party to break network security. Such an effort is easier to trace than a break-in by an outside source. However, a smart card can still be lost and used by someone who knows what the card is used for and who it belongs to (there's nothing to stop the user from placing some form of identification on the card, even if the issuing company doesn't). In addition, the issuing company has to bear the cost of issuing and maintaining the cards, which can be an expensive proposition.

There's a relatively new alternative to both passwords and smart cards. Biometrics is a statistical method of scanning people's unique characteristics to ensure that they are who they say they are. Some of the scanned elements include voiceprints, irises, fingerprints, hands, and facial features. The two most popular elements are irises and fingerprints because they're the two that people are most familiar. The advantages of using biometrics are obvious. Not only can't users lose their identifying information (at least not very easily), but with proper scanning techniques, the identifying information can't be compromised either.

At the time of this writing, there are two main problems with using biometrics: quality and prices. Some managers don't want to use biometrics because of the time it takes to create quality scans that ensure absolute accuracy. The quality issue is being resolved as computer hardware gets faster. In addition, the price for a single scanner can range from $100 for something simple to thousands of dollars for complex solutions. Considering the number of access points a typical company has, the price of using biometrics for all but the most stringent security requirements can be prohibitive. Obviously, you have to weigh the cost of the data you're trying to protect against the cost of protecting it.

Fortunately, the tide is turning for biometrics as more companies start to pursue this solution. Today, you can find biometrics in use in a wide range of government and high-security institutions. Many industry experts predict that forward thinking financial organizations will begin using biometrics as a means of identification by the end of 2000. By 2001, many corporations will view fingerprint identification as the access method of choice for remote communication. Iris identification may begin to overtake fingerprint identification as early as 2002, as the primary means of identification used by some corporations.

There are a few companies that deal with biometrics today. One of the most popular systems, the IriScan System 2100, can scan the iris of an individual using a camera instead of direct contact. The codes for the scanned individuals are stored in a Microsoft Access database. The system costs about $5,500 and requires you to provide a Windows 2000 4.0 or above workstation that has a minimum of a 166MHz Pentium processor, 32MB of RAM, and 1.7GB of hard drive space. You can find out more about IriScan, Inc. at `http://www.iriscan.com/index.htm`.

If your company prefers to use fingerprints instead of other biometric techniques to enforce security, you may want to look at the fingerprint reader chip solutions provided by Veridicom, Inc. (`http://www.veridicom.com/`). They're working with Intel to incorporate interfaces for biometrics into the Common Data Security Architecture (CDSA). CDSA is a comprehensive set of security services that will make secure transactions easier. It has a four-layer architecture: application, layered services and middleware, Common Security Services Manager (CSSM) infrastructure, and security service provider modules. You can find out more about CDSA at `http://www.opengroup.org/security/cdsa/index.htm`. CDSA is currently on The Open Group fast track to becoming a standard.

IriScan, Inc. and Veridicom, Inc. aren't the only companies getting into biometrics. Compaq computer has introduced a sub $100 fingerprint reader that it developed with Identicator Technology. You can find out about this and other Compaq biometric technologies at `http://www.compaq.com/newsroom/pr/1998/pr070798b.html`. A few other biometric product dealers include Saflink Corp. (`http://www.saflink.com/home.html`), Visionics, Corp. (`http://www.faceit.com/live/frameset.html`), Keyware Technologies (`http://www.keyware.be/products/vg_sdk.htm`), and Vision Systems, Inc. (`http://www.whovision.com/`). All of these companies provide various biometric products that can make your network just a bit more secure from prying eyes.

Summary

Security is an ongoing problem for any organization. You need to protect your data using a variety of methods, some of which may involve using new strategies that crackers haven't circumvented yet. In this chapter, we looked at many of the things you need to do to make your applications more secure. We also looked at some of the things you can do to monitor the security measures that you have in place. Here are some additional conclusions you can draw from the contents of this chapter.

- Security system testing is one of the only ways to ensure the best possible protection for your network. Ethical hackers provide the means to test the security of your network without the detrimental effects of an outside break-in.
- Most security functions can be divided into either the access token or security descriptor category. Although there are some functions that you need to use in both places, you can usually find a needed function faster by looking for the right function type.

- Both security descriptors and access tokens require functions that work with owner, group, SACL, and DACL information. In addition, some work with access tokens requires you to work directly with SIDs.
- The Access Control Editor allows you to design an object security interface that network administrators can quickly recognize with very little effort. The flexibility that this component provides allows you to create specialty displays such as user information dialog boxes.
- There are four main Windows security elements at work when the user logs into the system. Password filters ensure that the user provides a secure password that meets company password requirements. The Winlogon user interface (including GINA) allows the user to log onto the system using a graphical interface. The network provider API allows the system to verify the user's identity using network resources. The LSA authentication component works with GINA and other system elements to log the user into the system and issue an access token.
- Using passwords alone may not be enough to protect your data since passwords are prone to being lost or compromised. New technology solutions such as smart cards and biometrics can help you reduce the possibility of security breaches.

✦ ✦ ✦